Algorithms for Finding Dispensable Variables

Mikoláš Janota Joao Marques-Silva Radu Grigore

This short note reviews briefly three algorithms for finding the set of dispensable variables of a boolean formula. The presentation is light on proofs and heavy on intuitions.

It is sometimes desirable to find the set of variables that have value 0 in all minimal models of a boolean formula [2]. A *minimal model* is one in which flipping any variable's value from 1 to 0 leads to a non-model. All the models of the function $a \oplus b$ are minimal (01 and 10), and they are also the minimal models of $a \vee b$, which has one non-minimal model (11). For both these examples the set of dispensable variables does not contain variable a, nor variable b.

Preliminary definitions. A *literal* is a variable or the negation of a variable. A *clause* is a disjunction of literals, usually represented as a set. A *CNF* formula is a conjunction of clauses, usually represented as a list. A *model of a boolean formula* is a map from variables to values that makes the value of the formula 1.

1 The MaxSAT approach

A *weighted MaxSAT solver* takes as input a CNF formula with clauses $c_1, \ldots, c_m$ and positive weights $w_1, \ldots, w_m$ associated with each clause. The output is a model that maximizes $\sum_i w_i c_i$.

A weighted MaxSAT solver can be used to find a *cardinality minimum model*, a model that has as few variables with value 1 as possible. Suppose the original clauses are $c_1, \ldots, c_m$ and the variables are $v_1, \ldots, v_n$. We add the *clauses* $\neg v_1, \ldots, \neg v_n$, each with weight 1. We give to each of the original clauses weight $n + 1$. The weighted MaxSAT problem is now[1]:

$$\begin{array}{lccc} \text{clauses:} & c_1, \ldots, c_m, & \neg v_1, \ldots, \neg v_n & (1) \\ \text{weights:} & n+1, \ldots, n+1, & 1, \ldots, 1 & (2) \end{array}$$

It is easy to see that the weighted MaxSAT solver will satisfy $c_1, \ldots, c_m$ when it is possible, and will choose as many values of 0 for variables as possible.

[1] In fact, a non-weighted MaxSAT solver that knows about *hard* and *soft* clauses is enough.

A general approach. A cardinality minimum model is also a minimal model. (The converse is false.) Once we can find *one* minimal model we can generate *all* minimal models using the following algorithm:

GENERATE-MINIMAL-MODELS(f)

```
1  μ ← MINIMAL-MODEL(f)
2  while μ ≠ NIL
3        do VISIT(μ)
4           f ← f ∧ ⋀_{μ(v)} ¬v
5           μ ← MINIMAL-MODEL(f)
```

Note that the number of minimal models may be exponential in the number of variables and in the number of clauses:

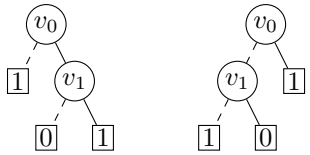
$$\bigwedge_{1\le k\le n} (v_{2k-1} \oplus v_{2k}) = (v_1 \oplus v_2) \wedge (v_3 \oplus v_4) \wedge \cdots \tag{3}$$

$$= (v_1 \vee v_2) \wedge (\neg v_1 \vee \neg v_2) \wedge (v_3 \vee v_4) \wedge (\neg v_3 \vee \neg v_4) \cdots \tag{4}$$

We are now investigating an approach that exploits the inner workings of a MaxSAT solver. In particular, some MaxSAT solvers use a *bound* on the solution value, and that evolves predictably when the formula is modified as in the previous algorithm.

2 The BDD approaches

Binary decision diagrams (BDD) are an alternative to CNF for representing boolean formulas. The function $\neg v_0 \vee v_1$ and the function $\neg v_1 \vee v_0$ have the following BDDs:



Even though the two functions are essentially the same, the BDDs are different because one constraint of BDDs is to have variables ordered on all paths from the root to a leaf. BDDs are directed acyclic graphs. To evaluate the formula for a given assignment of values to variables we start from the root and at each node look at the value of the variable that labels the node: If it is 0 then we take the *low* branch; if it is 1 we take the *high* branch. The value of the function is given by the leaf that is reached by this process. Another constraint on BDDs is that they have no redundant node: There is no node whose low and high branches point to the same place and there are no two nodes that have the same label and their respective branches point to the same place. (In particular, this 'no-redundancy' rule means we can't have two leafs with the same value, but that would be difficult to draw.)

Reusing the general approach. Certain operations are particularly easy to carry out on BDDs. For example we can find the *lexicographically minimum model* by starting at the root and always taking the low branch unless it leads to 0. A lexicographically minimum model is also a minimal model. (The converse is not true.) Therefore we can use the same approach as before and implement the procedure MINIMAL-MODEL using BDDs. Preliminary experiments show that MaxSAT solvers tend to work better in this context.

A BDD-specific solution. With BDDs we can:

1. Build a BDD that represents all *minimal* models.
2. Extract the set of dispensable variables from this BDD.

The function $minimal(f)$ gives the BDD whose all models are the minimal models of the BDD f.

$$minimal(v?h : l) = v?(minimal(h) \wedge monotone(l)) : minimal(l) \tag{5}$$

The notation $v?h : l$ denotes a BDD whose root node is labeled by variable v and whose high and low branches are pointing, respectively, to the BDDs h and l. A logical operation applied to two BDDs, such as $\wedge$ above, is understood to stand for the proper algorithm, which is outside the scope of this short note.

The function $monotone(f)$ gives a BDD whose models are all the models of f plus those that can be obtained by flipping the value 0 into value 1 for some variables in a model. For example, $monotone(a \oplus b) = a \vee b$. Interestingly, in this case *monotone* and *minimal* are inverses, since $minimal(a \vee b) = a \oplus b$.

$$monotone(v?h : l) = (v \wedge monotone(h)) \vee monotone(l) \tag{6}$$

Previously we did not discuss what procedure VISIT does to keep track of dispensable variables because it was obvious. But it is worth mentioning how the set of dispensable variables is obtained from $minimal(f)$. We can extract the set of variables that have the value 1 in some model as follows:

$$extract(v?0 : l) = extract(l) \tag{7}$$
$$extract(v?h : l) = \{v\} \cup extract(l) \cup extract(h) \tag{8}$$

The set of dispensable variables of a formula f is the complement of

$$extract(minimal(f)) \tag{9}$$

A few words about efficiency. Given BDDs f and g of sizes m and n it takes $O(mn)$ time (and space) to compute $f \circ g$ for any binary boolean operation $\circ$. But *typically* it takes only time proportional to $m + n$. As a result, a good (folklore to our knowledge) heuristic for going from CNF to BDD is to construct a small BDD for each clause, put them in a priority queue with the smallest at

its root, and then repetedly compute the binary operation $\wedge$ between the two smallest BDDs. (The problem of minimizing the time is the same as Huffman coding if time and space are both exactly $m + n$.)

Another interesting observation is that $|monotone(f)| \leq |f|$. Here $|f|$ denotes the size of the BDD representing the function f. (The result naturally extends to the *smallest* BDD under permutations of variables.) To understand why this is so it is useful to think of BDD nodes as being tagged with truth-tables [1]. For example, the truth-table of $a \oplus b$ is 0110 and it labels the root of the corresponding BDD. The low branch points to a node labeled by the first half 01 and the high branch points to 10. Therefore the nodes in the BDD of $a \oplus b$ are 0110, 01, 10, 0, and 1 for a total size of 5. On the other hand the truth-table of $a \vee b$ is 0111. The nodes in this case are 0111, 01, 0, and 1, for a total size of 4. Notice that there is no node 11. In fact there is never a square node (of the form aa for some a) because of the restriction that low and high branches are different. When we compute the *monotone* function the truth table lh becomes $l(l \vee h)$, where $\vee$ is bitwise: This operation (carried out recursively) may introduce square tables but may never remove them. Qed.

We are now exploring the relation between $|minimal(f)|$ and $|f|$. In our experiments it is almost always the case that $|miminal(f)| \leq |f|$, but we know this relation does not hold for $f = a \vee b$.

References

[1] Donald Erwin Knuth. *The Art of Computer Programming, Fascicle 1b: Binary Decision Diagrams*. Addison-Wesley Professional, March 2009.

[2] Mikoláš Janota, Goetz Botterweck, Radu Grigore, and Joao Marques-Silva. How to complete an interactive verification process? *International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, January 2010.